

Remarks

Applicants respectfully request reconsideration of this application as amended. Claims 21-24, 30-34, 40 and 41 have been amended. No claims have been canceled. Therefore, claims 21-45 are now presented for examination.

In the Office Action, claims 21, 22, 31, 32, 41 & 42 stand rejected under 35 U.S.C. 102 (e) as being anticipated by Hollander, U.S. Patent No. 6,347,388 B1 ("Hollander").

Applicants submit that the present claims are patentable over Hollander. Hollander discloses a test generator module for automatically creating a device verification test from a functional description. See Hollander at Figure 1. The test generator can be constrained to generate tests for specific subsets of the design's functionality. Thus, some tests can focus on a specific feature in the design, while other tests can address broader functional scopes (col. 4, ll. 66 – col. 5, ll. 5).

In addition, data checks are performed to verify the relation among different data elements, or to verify data against a high level reference model. The generator can perform any combination of static and dynamic checks. When using both dynamic generation and dynamic checking, the test generator module and the checker can constantly synchronize. Thus, temporal checks can be precisely targeted (col. 5, ll. 18-25). Further, reports of functional coverage statistics at the architectural, module, or system levels are provided. Cross coverage reports can also be generated for later analysis (col. 5, ll. 30-40).

Claim 21 recites generating a second program if an IC has not been sufficiently tested by a first test program, wherein the second test program includes an updated test program population having a second set of instructions and data being a mutation of the original population. Applicant submits that nowhere in Hollander is there disclosed generating a second program including an updated test program population having a second set of instructions and data being a mutation of an original population if an IC has

not been sufficiently tested by a first test program. Therefore, claim 21 is patentable over Hollander.

Claims 22-30 depend from claim 21 and include additional limitations.

Therefore, claims 22-30 are also patentable over Hollander.

Claim 31 recites causing a processor to generate a second program if the IC has not been sufficiently tested by the first test program, the second test program including an updated test program population having a second set of instructions and data being a mutation of the original population. For the reasons stated above with respect to claim 21, claim 31 is also patentable over Hollander. Since claims 32-40 depend from claim 31 and include additional limitations, claims 32-40 are also patentable over Hollander.

Claim 41 recites a feedback engine to build and update a population of test programs by generating an abstract syntax tree (AST) for each test program. Applicants submit that Hollander does not disclose such a limitation. Thus, claim 41 is also patentable over Hollander. Because 42-45 depend from claim 41 and include additional limitations, claims 42-45 are also patentable over Hollander.

Claims 23, 24, 33 & 34 stand rejected under 35 U.S.C. 103 (a) as being unpatentable over Hollander, in view of Hayes, U.S. Patent No. 5,799,266 ("Hayes"). Applicants submit that the present claims are patentable over Hollander even in view of Hayes.

Hayes discloses a method and apparatus for designating sequences of interrelated test functions of software interfaces and specifying selected attribute values of the test functions' parameter attributes, and automatically generating from these designations/specifications. See Hayes at col. 2, ll. 19-25. A test driver generator is provided for generating test drivers. The test driver generator receives test expressions designating execution sequence of test functions of software interfaces and corresponding attribute value specifications for the designated test functions' parameter attributes. Each test expression designates a number of test functions to be executed in a certain sequence,

and each corresponding attribute value specification specifies selected attribute values of the test functions' parameter attributes.

For each test expression and corresponding attribute value specifications of a software interface, the test driver generator, in response, generates a test driver that can execute the specified test functions in the designated order with all combinations of the selected attribute values of the test functions' parameter attributes (col. 2, ll. 35-49). Further, a parser parses and tokenizes test function designations and attribute value specifications based on formal grammar. The intermediate representation builder builds a syntax tree. The code generator generates the test drivers using the syntax tree.

Nevertheless, Hayes does not disclose or suggest generating a second program including an updated test program population having a second set of instructions and data being a mutation of an original population if an IC has not been sufficiently tested by a first test program. Moreover, Hayes does not disclose or suggest a feedback engine to build and update a population of test programs by generating an abstract syntax tree (AST) for each test program. As described above, Hollander does not disclose or suggest such limitations. Therefore, any combination of Hollander and Hayes would also not disclose or suggest such limitations. Consequently, the present claims are patentable over any combination of Hollander and Hayes.

Claims 25-30, 36-40 & 43-45 stand rejected under 35 U.S.C. 103 (a) as being unpatentable over Hollander, in view of Hayes, and further in view of Miller et al., U.S. Patent No. 6,175,948 B1 ("Miller"). Applicants submit that the present claims are patentable over Hollander and Hayes even in view of Miller.

Miller discloses a process wherein source code for a component is extracted from a database and parsed into an AST. However, Miller does not disclose or suggest generating a second program including an updated test program population having a second set of instructions and data being a mutation of an original population if an IC has not been sufficiently tested by a first test program. Additionally, Miller does not disclose

or suggest a feedback engine to build and update a population of test programs by generating an abstract syntax tree (AST) for each test program. As described above, Hollander and Hayes do not disclose or suggest such limitations. Therefore, any combination of Hollander, Hayes and Miller would also not disclose or suggest such limitations. Accordingly, the present claims are patentable over any combination of Hollander, Hayes and Miller.

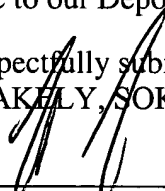
Applicants respectfully submit that the rejections have been overcome, and that the claims are in condition for allowance. Accordingly, applicants respectfully request the rejections be withdrawn and the claims be allowed.

The Examiner is requested to call the undersigned at (303) 740-1980 if there remains any issue with allowance of the case.

Please charge any shortage to our Deposit Account No. 02-2666.

Date: 1/2/03

Respectfully submitted,
BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN LLP



Mark L. Watson
Reg. No. 46,322

12400 Wilshire Boulevard
7th Floor
Los Angeles, California 90025-1026
(303) 740-1980

Version with Markings to Show Changes Made
Insertions are underlined, deletions are bracketed.

1 21. (Amended) A method comprising:
2 generating a first test program to test the functionality of an integrated circuit
3 (IC), the first test program including a test program population having a first set of
4 instructions and data;
5 executing the first test program;
6 [determining whether] evaluating a first set of coverage data from the first test
7 program to determine if the IC has been sufficiently tested, wherein evaluating the first
8 set of coverage data comprises comparing the coverage data to a predetermined coverage
9 requirement; and
10 generating a second program if the IC has not been sufficiently tested by the first
11 test program, the second test program including an updated test program population
12 having a second set of instructions and data being a mutation of the original population.
13 [if not, determining whether a predetermined test program population threshold has been
14 reached.]

1 22. (Amended) The method of claim 21, further comprising:
2 [generating a second test program if the predetermined test program population
3 threshold has been reached; and]
4 executing the second test program.

1 23. (Amended) The method of claim 22, wherein generating the first test program
2 comprises:
3 generating a first abstract syntax tree (AST);
4 generating [a] the first set of instructions and data for the first AST; and
5 translating the first AST into a first executable test program.

1 24. (Amended) The method of claim 23, wherein generating the second test
2 program comprises:

3 generating a second abstract syntax tree (AST);

4 generating [a] the second set of instructions and data for the second AST; and

5 translating the second AST into a second executable test program.

1 30. (Amended) The method of claim [21] 23, further comprising:

2 adding the first AST and [corresponding] the first set of coverage data into test

3 program population after the first test program has been executed.

1 31. (Amended) A computer system comprising:

2 a storage device coupled to a processor and having stored therein at least one

3 routine, which when executed by the processor, causes the processor to generate data, the

4 routine causing the processor to,

5 [generate a first test program to test the functionality of an integrated circuit (IC);

6 execute the first test program;

7 determine whether the IC has been sufficiently tested; and

8 determine whether a predetermined test program population threshold has been

9 reached.]

10 generate a first test program to test the functionality of an integrated circuit (IC),

11 the first test program including a test program population having a first set of instructions

12 and data;

13 execute the first test program;

14 evaluate a first set of coverage data from the first test program to determine if the

15 IC has been sufficiently tested, wherein evaluating the first set of coverage data

16 comprises comparing the coverage data to a predetermined coverage requirement; and

17 generate a second program if the IC has not been sufficiently tested by the first
18 test program, the second test program including an updated test program population
19 having a second set of instructions and data being a mutation of the original population.

1 32. (Amended) The computer system of claim 31, wherein the routine further
2 causes the processor to,
3 [generate a second test program if the predetermined test program population
4 threshold has been reached; and]
5 execute the second test program.

1 33. (Amended) The computer system of claim 32, wherein generating the first test
2 program comprises:
3 generating a first abstract syntax tree (AST);
4 generating [a] the first set of instructions and data for the first AST; and
5 translating the first AST into a first executable test program.

1 34. (Amended) The computer system of claim 33, wherein generating the second
2 test program comprises:
3 generating a second abstract syntax tree (AST);
4 generating [a] the second set of instructions and data for the second AST; and
5 translating the second AST into a second executable test program.

1 40. (Amended) The computer system of claim [31] 33, wherein the routine further
2 causes the processor to,
3 [adding] add the first AST and [corresponding] the first set of coverage data into
4 test program population after the first test program has been executed.

1 41. (Amended) A validation test system comprising:

2 a test builder to generate test programs to test the functionality of an integrated
3 circuit (IC);
4 a test generator to translate the test programs into an executable test;
5 a test analyzer to execute the test programs; and
6 a feedback engine to build and update a [test] population of test programs by
7 generating an abstract syntax tree (AST) for each test program.